

Microsoft Research Short Course

**COM+ Runtime Technology
(Part Two)**

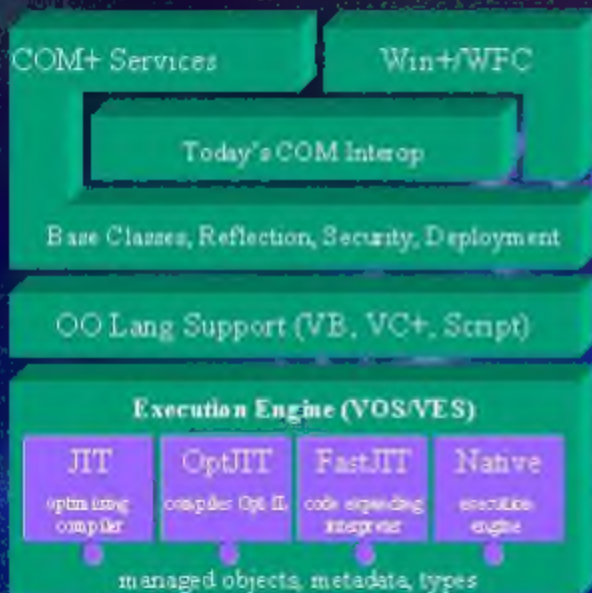
Jim Miller

**Program Manager,
COM+ Runtime**

COM+ Short Courses

- **Last Week: “VOS level”**
- **Today: “Enterprise”**
 - ◆ Garbage Collection
 - ◆ Assemblies
 - ◆ Contexts
 - ◆ Remoting
- **Proposed: Security**
- **Proposed: What's wrong with the COM+ Runtime**

Architectural Overview



- Lowest level is Microsoft IL + objects
- Language support is built over execution and object model
- Some languages assume reflection, base classes
- COM classic interop assumes classes, reflection, deployment
- COM+ Services are combination of COM classic and runtime objects
- WFC and Win+ will leverage COM classic interop

Agenda

- **Garbage Collection**
- Contexts
- Assemblies
- Remoting

Garbage Collection

- Lifetime of object based on reachability
- Starts from roots: Stack and globals.



FOR MORE INFO...

Garbage Collection: Richard Jones and Rafael Lins (Wiley)

Uniprocessor Garbage Collection Techniques: Paul Wilson
<http://www.cs.utexas.edu/users/oops/papers.html>

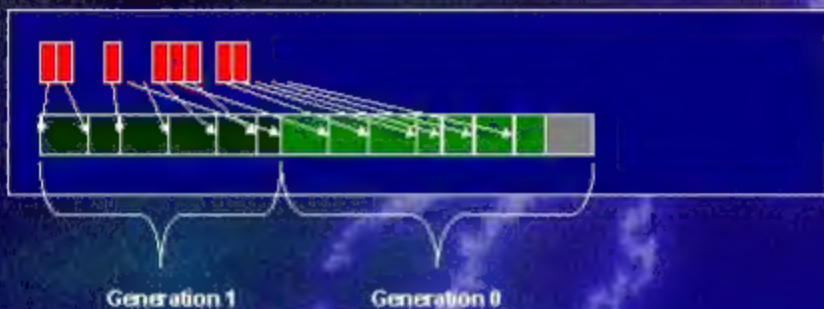
COM+ EE Collector

- Mark and Compact Collector
- Generations with write barrier
- Keeps objects ordered
- Low fragmentation, low memory overhead
- Cache conscious
- No conservative GC support
- Pinning

High Performance Allocator

- Automatic Memory Management
 - ◆ State of the Art Garbage Collected Core
 - Generational compacting
 - Low Fragmentation / Memory Usage
 - ◆ Transient Activity / Transaction Heaps
- Performance Results/Expectations (P200)
 - ◆ 1ms Max Pause for Gen 0
 - ◆ < 10ms Max Pause (w/ Write Watch OS Support)
 - ◆ Allocate/Lifetime/Free Competitive to Heap Alloc
 - ◆ Collection as Intrusive as Page Fault
- Allocation Contexts
 - ◆ No locks
 - ◆ No cache snooping

Generational GC



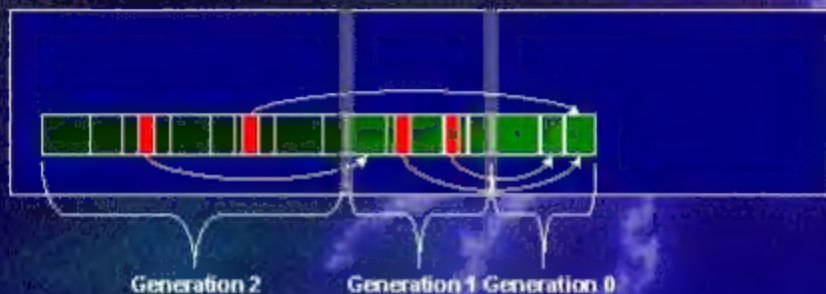
New Heap Begins with New Generation
Accessible References Keep Objects Alive
Preserves / Compacts Referenced Objects
Objects Left Merge with Older Generation
New Allocations Rebuild New Generation

Generational GC



- Generations Dynamically Tuned
 - CPU Cache Size
 - Acceptable Fragmentation
- Older Generations are Larger / More Stable
 - Require Collection Less Often
 - Are More Expensive to Collect
 - Can Have References to Newer Objects

Write Barrier Support



- **Cross Generation (old->new) Refs Tracked**
- **Great Performance / Working Set Benefits**
- **OS Cooperation Improves Both (WriteWatch)**
 - Enables Concurrency / Maximum Pause < 10ms
 - Ability to Minimize Working Set Impact

Dynamic Tuning

- Decides when and how to GC
- Gen 0 should fit in L2 cache
- Constant growth model
- Estimates when is a good time to GC
- Chooses sweep or compact based on fragmentation limits, set for each generation

We win because:

- Ephemeral design
- Cache conscious approach
 - Gen 0 fits in L2
 - Only Mark phase is "random walk"
- Respects object placement
- Low memory overhead

GC Scaling Performance

```

class Small
{
    int i, j;

    public Small(int i, int j)
    { super(i, j); }
}

for (int iter = 1; iter < 1000000000; iter++)
{
    Small s = new Small(12, 34);
    if (s.i % 2 == 0 || s.j % 3 == 0) throw
        new RuntimeException("Small");
}

// threads: tested as multiprocessor, 14 cycles/iteration
// threads: multiprocessor, 58
// threads: multiprocessor, 66

```

Agenda

- Garbage Collection
- Contexts
- Assemblies
- Remoting

Contexts

- Generalized support for object isolation
- Processes are partitioned into contexts
- Each thread has a current context
- Context represented by an object
- Policies control inter-context references, based on:
 - Full process stack at call site
 - Method and parameters (or field)
 - Destination object and class

Contexts

Context1



Context2



Policy Examples

• Access Checking

• User Identity

• Synchronization

• Transaction boundary

Examples of Contexts

- Application Domain
 - ◆ Class Identity not preserved
- Cross Process
 - ◆ Copy In/Copy Out, not sharing
- Cross Machine
 - ◆ Like Cross Process, but copying may be slow and error prone
- Split Classes
 - ◆ Think of them as in all contexts within a given process, simultaneously

Cross Context Communication

Proxy

Link

Dispatcher

- Policy Enforcement
- Control transfer
- Data transfer

Agenda

- Garbage Collection
- Contexts
- Assembly
- Remoting

The Deployment Problem

- **Fragile**

- I installed my new database modeling package, now all apps that use OLE DB are broken and need some a clue what's wrong.

- **Complex**

- Why do I have to understand registry again in order to figure out if my application is correctly installed?

- **Opaque**

- What will break if I delete this directory?
- Do I have everything I need to run this app off line?
- What do I need to move the app to my new machine?
- How did this get here?

The Solution

Application Isolation

- New applications don't break old applications when new versions of components are installed.
- Configuration rules for an application affect behavior of all components acting on it, even if they are shared.
- Sharing is explicit.

Simple Registration Model

- No registry dependency.
- Self-describing applications.

Runtime Infrastructure

- An integrated set of services to define, package, advertise, manage, and monitor component-based applications.

Building Block Assemblies



- A collection of resources (executable files, etc.) that are built to work together in "logical" dll.
- The fundamental unit of deployment, versioning and reuse
 - Private mapping of logical resource to physical packaging
 - Manifest describing
 - Top level list of physical resources
 - Top level dependencies on other assemblies
- Examples: MDAC, Word, a custom or Banking application, WFC

Manifests



- Manifest defines policy and assembly structure
 - Locating referenced assemblies
 - Enforcing version binding rules
 - Additional configuration rules
- Authored or emitted by tool, edited by admin
 - Can be inside or outside of component metadata
 - Can be stored alone or as a packaging decision
 - Admin can configure, but original retained as "safe mode"
- Not required for simple deployment

Assemblies and Development

- Always author in an assembly
- Assembly establishes development-time namespace
- Extend the namespace by importing other assemblies

```
#import com.abcorp.bank as bank  
  
c = bank.CAccount()
```



AssemblyRef

com.abcorp.bank

Unlabeled, x86, German, Mono

Assemblies and Deployment

- Deployment can be as simple as xcopy
 - ◆ "Run From Source"
 - ◆ Manifest not required
- Integrated with:
 - ◆ MSI
 - ◆ ZAW
 - ◆ SMS, etc...
- On Demand, Zero Impact Installation

Assemblies at Runtime:

Isolation

- Isolation of code, data, settings,
- Essential for solving DLL Hell
- Assemblies establish namespace for resolution
- Manifest governs how references are resolved
- Side by Side
 - Multiple versions of an assembly running in the same process
- Application Domains

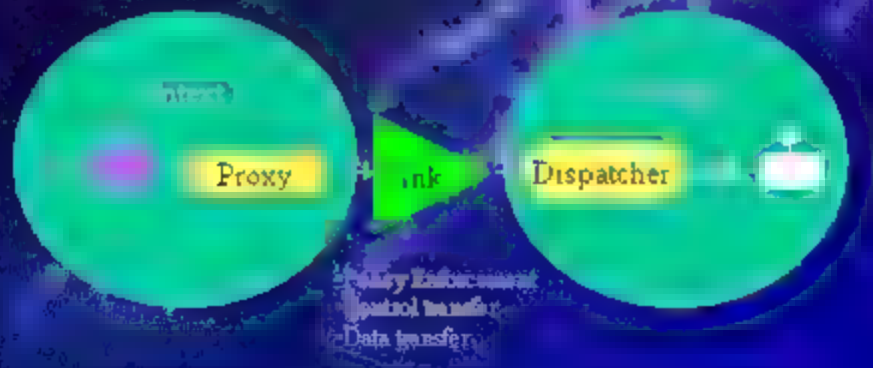
Versioning

- Versions expressed at the Assembly level – not the class level
- Editable by administrators
 - Original manifest immutable; changing policy implies creating a “new manifest”
 - Use Latest
 - Use specific version
 - Use the version I was built with
- Safe Mode
 - Hashes recorded for every reference
 - “Run Once, Run Forever”

Agenda

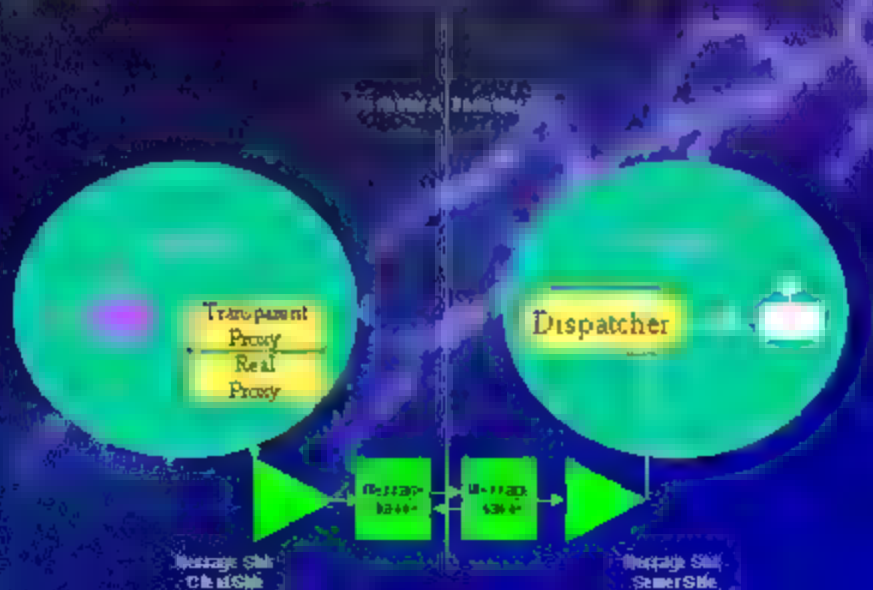
- Garbage Collection
- Contexts
- Assemblies
- **Remoting**

Remoting Based on Contexts



Remoting Based on Contexts

Arkadiusz Wójcik



Message Sinks

- Support for request-response, one-way messages, multicast events
- Composable for complex scenarios
 - sequences represent logical driver layers
 - fan-out, fan-in and pull (stream/DB based)
- Optionally Incorporate Key Properties
 - object identity
 - [security] identity information
- Optional Reply sink on Send

From Interfaces to Objects

- COM: interfaces and basic data types
- Smalltalk, VC, Java, C++OL: "Objects"
 - ◆ Interfaces
 - ◆ Classes
 - Methods
 - Fields
 - Statics

Remoting

- Runtime generates proxies and stubs automatically
- Features
 - ◆ Standard set of remotable types (CLS)
 - ◆ Metadata allows us to remote self describing types
 - ◆ Objects passed by Value or by Reference

Properties and Events

■ Properties

- ◆ As in COM
- ◆ Users think they are fields
- ◆ Tools implement them with methods

■ Events

- ◆ One-way communication
- ◆ Loosely coupled
- ◆ “Plumbing” with “register” and “fire”
- ◆ Events pass data when they fire, captured as a signature on the event

Remoting Programming Model

Subject to change without notice

- Programmers should assume all classes support remote instances.
- The current object is local, including all inherited state (this and super).
- All fields are in current context.
- Field access to remote objects is remoted.
- Static access to remote classes (fields or methods) generates a JIT time exception.
- Recall that instances of agile classes are available in all contexts.

Remoting Programming Model (A Thought Experiment)

```
Class Complex
{
    double R, Theta;
}

Complex Matrix[]
    new {Complex[10000, 10000],
        ...
        ... ("supercomputer.org")};

Initialize(Matrix)
Matrix.Invert()
Text.Out.WriteLine(Matrix[0].Theta)
```

Cross Process Remoting

- **LPC Sink** - Uses channel to send and receive messages
- **LPC Manager**
 - Initializes cross process remoting
 - Creates and deletes channels
- **LPC Channel**
 - Manages LPC Ports
 - Managed memory sections
 - Send and Receive Messages

Cross Machine Remoting

- Simple Object Access Protocol (SOAP)
- SOAP *is* COM+ remoting over HTTP.
- Outreach: Simple enough that SOAP endpoints can easily be created using a wide variety of technologies, including scripts
 - Perl, Python, ???, .. (ad hoc)

SOAP Goals

Interaction across the web with these goals:

- **Work well with existing Web infrastructure**
- **Work better with future Web infrastructure**
- **Introduce no new security holes**
- **Support reach requirements of partners**
- **Lower barriers to adopting COM+ runtime**
 - ◆ enable interop solutions in heterogeneous environments
 - ◆ including classic COM

SOAP Method Invocation

- Objects represented by URL
 - ◆ Well known endpoints: no ORB needed
 - ◆ Dynamically create instances
- Object invocation over HTTP via M-POST, POST, and GET
- Marshalling using XML for interoperability
 - ◆ optional interface
 - ◆ Method
 - ◆ Parameters

Open Issues

- **Activation**
- **Programming Model**
- **Distributed Security**
- **Concurrent GC**
- **Distributed GC**